Tugas Kecil 1 Strategi Algoritma
Algoritma Brute Force dalam Penyelesaian Linkedin Queens Puzzle

Moh. Hafizh Irham Perdana (13524025)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13524025@std.stei.itb.ac.id, hafizhirham06@gmail.com

I. PENDAHULUAN

Linkedin Queens Puzzle adalah teka-teki visual dan logika dengan premis yaitu kita perlu mengisi kotak-kotak yang tersedia dengan Queen sedemikian sehingga hanya terdapat satu Queen di setiap baris, kolom, dan wilayah warna tanpa terdapat Queen yang bersentuhan bahkan secara diagonal. Teka-teki ini mengasah strategi dan intuisi kita dengan memahami parameter-parameter petunjuk, seperti warna mana yang memiliki daerah paling kecil sehingga memiliki kemungkinan benar yang lebih besar.

Teka-teki tersebut dapat diselesaikan dengan berbagai metode dan algoritma salah satunya Algoritma Brute Force. Pendekatan ini digunakan agar kita dapat mengetahui bagaimana algoritma *brute force* bekerja dalam kasus mengenumerasikan posisi Queen pada papan *grid* dengan dimensi $N \times N$.

II. DASAR TEORI

A. Algoritma Brute Force

Algoritma *brute force* adalah algoritma dengan menyelesaikan masalah secara lempang, langsung, dan sederhana tanpa memperhatikan pola-pola tertentu. Karena kelurusannya tersebut algoritma ini dapat menjamin solusi untuk setiap permasalahan dan dapat digunakan di persoalan apapun karena tidak terkait dengan aturan-aturan formal tertentu.

Namun, algoritma ini juga memiliki kekurangan yang cukup fatal apabila digunakan pada kasus besar dengan jumlah komputasi yang banyak karena benar-benar mengecek seluruh kemungkinan. Algoritma *brute force* dapat mempunyai kompleksitas hingga O(n^n) tanpa optimasi tertentu sehingga memakan terlalu banyak waktu dan sumber daya pada kasus besar. Melihat kekurangan tersebut, *brute force* hanya efektif dilakukan pada kasus dengan parameter kecil hingga menengah.

Contoh algoritma brute force adalah membandingkan setiap elemen pada *list* atau senarai untuk mencari elemen terbesar, membandingkan setiap isi senarai untuk melakukan pencarian(*searching*), atau mencoba seluruh kemungkinan untuk menyelesaikan teka-teki Sudoku.

B. Exhaustive Search

Exhaustive Search adalah istilah spesifik untuk algoritma Brute Force yang menyelesaikan permasalahan dengan mencoba semua kemungkinan yang ada satu per satu sampai ditemukan solusi yang benar. Algoritma ini banyak digunakan untuk memecahkan persoalan-persoalan kombinatorika yang kemungkinan solusinya merupakan permutes, kombinasi, atau himpunan bagian dari sekumpulan objek-objek diskrit. Meskipun *exhaustive search* secara teoretis menghasilkan solusi, namun waktu dan sumber daya yang digunakan sangat besar seperti algoritma *brute force* pada umumnya.

Contoh persoalan yang dapat diselesaikan menggunakan *exhaustive search* adalah Travelling Salesperson Problem(TSP). TSP adalah persoalan seorang pedagang yang harus pergi melalui setiap kota tepat hanya sekali dengan total jarak terpendek dan kembali lagi ke kota asal keberangkatan. Menggunakan *exhaustive search* artinya mengenumerasi semua kemungkinan rute yang dapat dilalui kemudian diambil solusi rute dengan total jarak terpendek.

Contoh lain penggunaan *exhaustive search* adalah penyelesaian masalah Knapsack, yaitu apabila diberikan n buah objek dan sebuah *knapsack* dengan kapasitas bobot K dengan setiap objek memiliki bobot w dan keuntungan p, bagaimana cara terbaik memilih objek-objek tersebut agar dapat masuk ke dalam *knapsack* sehingga mendapatkan keuntungan maksimal.

III. ALGORITMA

A. Tipe Data

Dalam implementasi program, struktur data didefinisikan menggunakan paradigma *Object Oriented*. Namun, untuk keperluan perancangan algoritma, struktur tersebut dapat direpresentasikan sebagai Tipe Bentukan (ADT) yang terdiri dari ukuran papan N, matriks warna, dan array Queens.

Ukuran papan bersifat dinamis dan ditentukan saat *runtime* berdasarkan input file, sehingga struktur data array dialokasikan sesuai ukuran N tersebut.

```
type Board : <
    N       : integer,
    Grid    : array [0..N-1, 0..N-1] of char,
    Queens  : array [0..N-1] of integer
    >
```

```
{ Catatan: Queens[i] = j berarti Ratu pada baris ke-i terletak di kolom ke-j }
```

B. Validasi

Sebelum menempatkan ratu pada posisi tertentu, algoritma harus memvalidasi apakah posisi tersebut aman. Fungsi IsSafe memeriksa empat batasan utama sesuai deskripsi persoalan:

- Batasan Kolom: Tidak ada ratu lain di kolom yang sama.
- Batasan Warna (Region): Tidak ada ratu lain yang menempati warna wilayah yang sama.
- Batasan Sentuhan (Touching Rule): Ratu tidak boleh bersentuhan dengan ratu lain secara vertikal, horizontal, maupun diagonal.
- Batasan Baris: Terpenuhi secara implisit oleh struktur data ArraySolusi.

```
function IsSafe(input B: Board, input row: integer, input col: integer) -> boolean
{ Mengembalikan true jika posisi (row, col) aman untuk ditempati Ratu }

KAMUS LOKAL
   r, c : integer
   Color : character
   distRow, distCol : integer

ALGORITMA
   Color <- B.Grid[row][col]

   for (r <- 0 to row-1) do
      c <- B.Queens[r]

      if (c = col) then
         return false

      if (B.Grid[r][c] = Color) then
         return false

      distRow <- |row - r|
      distCol <- |col - c|

      if (distRow <= 1) and (distCol <= 1) then
         return false

   return true
```

C. Pencarian Solusi

Pencarian solusi menggunakan pendekatan *Backtracking*. Algoritma mencoba menempatkan Queen di setiap baris. Kemudian program akan mengecek validasi sesuai kolom, *region*, dan *adjacency*. Apabila valid maka program akan melakukan rekursi namun jika tidak ada kolom yang aman di baris saat ini, program akan mundur (*backtrack*) ke baris sebelumnya dan mencoba kolom lain. Fungsi ini bersifat rekursif dan mengembalikan true ketika satu solusi valid ditemukan.

Program ini termasuk ke adalam algoritma *brute force* karena ia mencari seluruh kemungkinan sesuai dengan batasan atau syarat dalam Queens Puzzle.

```
function Solve(input/output B: Board, input row: integer) -> boolean
{ Menghasilkan true jika solusi ditemukan dengan algoritma Brute Force }

KAMUS LOKAL
   col : integer

ALGORITMA
   N <- getSize(B)
   if (row = N) then
       isValid <- CheckEntireBoard(B)

   if (isValid) then
       return true
   else
       return false

   for (col <- 0 to N-1) do
       B.Queens[row] <- col
       UpdateGUI(B, false)

       if (Solve(B, row + 1)) then
          return true

       B.Queens[row] <- -1

   return false
```

D. Kelas dan Fungsi

TABEL 1. DAFTAR KELAS DAN FUNGSI

Kelas	Fungsi	Input	Output
IOUtils	`readBoard`	String filename (Path file)	Objek Board
	`saveSolution`	String filename, Board board, long timeMs, long iterations	void (Menghasilkan file .txt)
Board	`Board`	char[][] grid	Instance dari kelas Board
	`getSize`	-	int (Ukuran board N)
	`getColor`	int r, int c (Koordinat)	char (Kode warna sel)
	`setQueen`	int row, int col	void
	`getQueenCol`	int row	int (Kolom Queen)
	`isSafe`	int row, int col	boolean
	`printBoard`	-	void (Cetak board)
	`toString`	-	String (Representasi board)
Solver	`Solver`	int initialDelay	Instance dari kelas Solver
	`setDelay`	int delayMs	void
	`setListener`	SolverListener listener	void
	`getIterations`	-	long
	`solve`	Board board, int row	boolean

	checkEntireBoard	Board board	boolean
	updateGUI	Board board, boolean forceUpdate	void
Board Panel	setBoard	Board board	void
	generateColors	-	void (Mengisi colorMap)
	paintComponent	Graphics g	void (Menggambar board ke layar)
	drawCrown	Graphics2D g2d, int x, y, size	void (Menggambar Queen)
	drawEmptyState	Graphics g	void
Queens GUI	QueensGUI	-	Instance JFrame (Window GUI)
Main	main	String[] args	void (Titik awal program)

Adapun kegunaan masing-masing dari fungsi dan prosedur tersebut adalah sebagai berikut,

1. Kelas IOUtils
   - `readBoard`: Berfungsi untuk membaca file teks eksternal, memvalidasi format, dan mengonversinya menjadi objek Board.
   - `saveSolution`: Digunakan untuk menyimpan hasil akhir Board yang telah terpecahkan ke dalam file teks statistik waktu dan jumlah iterasi.
2. Kelas Board
   - `Board` (*Constructor*): Melakukan inisialisasi objek papan berdasarkan grid warna yang diberikan dan menyiapkan array posisi Queen.
   - `getSize`: Mengambil nilai dimensi atau ukuran papan.
   - `getColor`: Mendapatkan kode karakter warna pada koordinat tertentu.
   - `setQueen`: Menempatkan atau menghapus posisi ratu pada baris tertentu pada array.
   - `getQueenCol`: Mengembalikan indeks kolom Queen.
   - `isSafe`: Mengecek validitas penempatan Queen berdasarkan aturan kolom, kesamaan warna, dan persinggungan antar sel.
   - `printBoard`: Mencetak representasi visual papan dalam bentuk karakter sederhana ke terminal/konsol.
   - `toString`: Mengonversi status Board saat ini menjadi sebuah string untuk keperluan penyimpanan file.
3. Kelas Solver
   - `Solver` (Constructor): Inisialisasi `solver` dengan pengaturan jeda waktu (delay) tertentu.
   - `setDelay`: Mengatur kecepatan *live update*.
   - `setListener`: Menghubungkan logika `solver` dengan antarmuka (GUI).
   - `getIterations`: Mengambil total jumlah iterasi yang telah dilakukan.
   - `solve`: Implementasi algoritma backtracking secara rekursif.
   - `checkEntireBoard`: Melakukan validasi seluruh Board.
   - `updateGUI`: Melakukan *update* GUI pada iterasi tertentu.
4. Kelas BoardPanel
   - `setBoard`: Memasukkan data Board ke dalam GUI.
   - `generateColors`: Memetakan setiap kode karakter unik pada *grid* sesuai warna.
   - `paintComponent`: Menggambar *grid*, warna kotak, garis batas, dan Queen ke layar.
   - `drawCrown`: Menggambar bentuk Queen.
   - `drawEmptyState`: Menampilkan pesan teks instruksi di tengah layar pada *initial state.*
5. Kelas QueensGUI & Main
   - `QueensGUI` (Constructor): Membuat seluruh komponen GUI seperti tombol, slider, dan letak.
   - `main`: *Entry point* program yang menjalankan aplikasi dalam thread Swing.

IV. Implementasi Program

A. *Kelas IOUtils*

Program ini dijalankan pada awal proses untuk membaca input .txt, melakukan validasi ukuran *board*, dan menyimpan masing masing komponen *board* ke dalam tipe data Board berupa *array* dua dimensi yang menyimpan koordinat kolom dan baris untuk setiap warna.

Kemudian `saveSolution` diimplementasikan dengan menulis Board solusi ke dalam .txt dengan statistik ukuran, waktu, dan jumlah iterasi yang dibungkus dengan *try* & *catch* untuk mencegah error.

```
public static void saveSolution(String filename, Board board, long timeMs, long iterations) {
    try (FileWriter writer = new FileWriter(filename)) {
        writer.write(board.toString());
        writer.write("\nUkuran board: " + board.getSize() + "x" + board.getSize() + "\n");
        writer.write("Waktu pencarian: " + timeMs + " ms\n");
        writer.write("Banyak kasus yang ditinjau: " + iterations + " kasus\n");
        System.out.println("Solusi berhasil disimpan di: " + filename);
    } catch (IOException e) {
        System.out.println("Gagal menyimpan solusi: " + e.getMessage());
    }
}
```

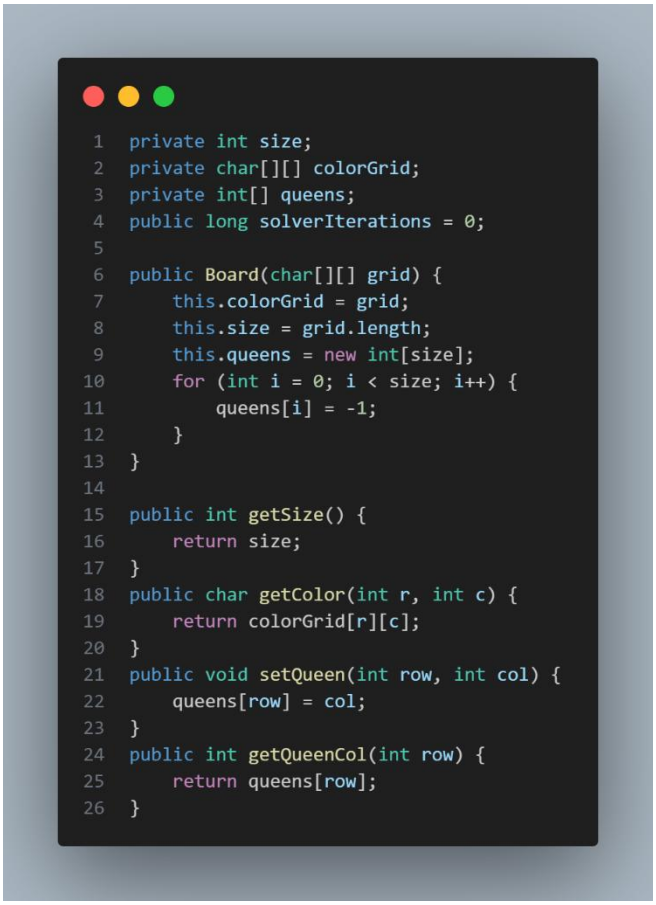
Gambar 1. Implementasi saveSolution

```java
public static Board readBoard(String filename) throws IOException {
        File file = new File(filename);
        if (!file.exists()) {
            throw new IOException("File tidak ditemukan: " + filename);
        }

        List<String> lines = new ArrayList<>();
        try (BufferedReader br = new BufferedReader(new FileReader(file))) {
            String line;
            while ((line = br.readLine()) != null) {
                line = line.trim();
                if (!line.isEmpty()) {
                    lines.add(line);
                }
            }
        }

        if (lines.isEmpty()) {
            throw new IOException("File kosong!");
        }

        int rows = lines.size();
        int cols = lines.get(0).length();

        if (rows != cols) {
            System.out.println("Peringatan: Input tidak persegi (" + rows + "x" + cols + ").");
        }

        char[][] grid = new char[rows][cols];
        for (int i = 0; i < rows; i++) {
            String rowStr = lines.get(i);
            if (rowStr.length() != cols) {
                throw new IOException("Panjang baris ke-" + (i + 1) + " tidak konsisten.");
            }
            grid[i] = rowStr.toCharArray();
        }

        return new Board(grid);
    }
```

Gambar 2. Implementasi readBoard

B. Kelas Board

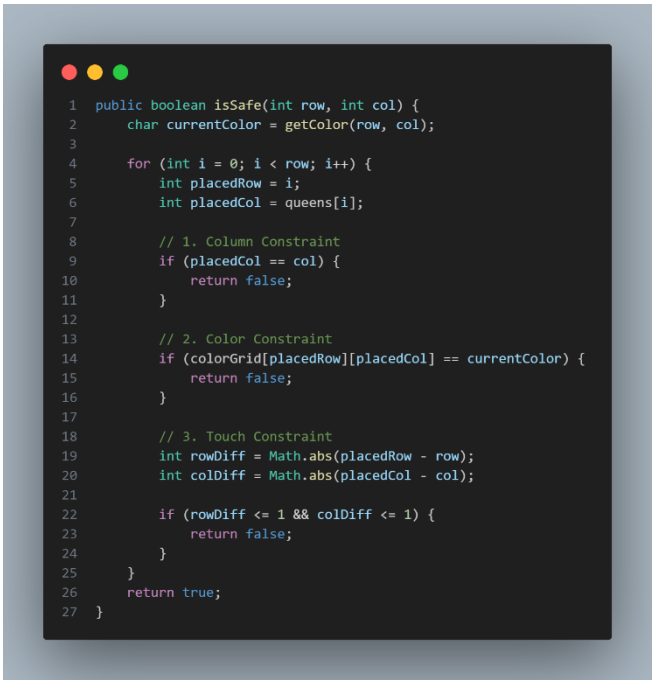
Program ini digunakan untuk menyediakan kebutuhan atribut untuk *board*. Terdapat metode untuk menginstansiasi *board*, mengambil ukuran *board*, mengambil warna pada koordinat tertentu, meletakkan Queen, dan mengambil kolom Queen.



```java
private int size;
private char[][] colorGrid;
private int[] queens;
public long solverIterations = 0;

public Board(char[][] grid) {
    this.colorGrid = grid;
    this.size = grid.length;
    this.queens = new int[size];
    for (int i = 0; i < size; i++) {
        queens[i] = -1;
    }
}

public int getSize() {
    return size;
}
public char getColor(int r, int c) {
    return colorGrid[r][c];
}
public void setQueen(int row, int col) {
    queens[row] = col;
}
public int getQueenCol(int row) {
    return queens[row];
}
```

Gambar 3. Implementasi Board

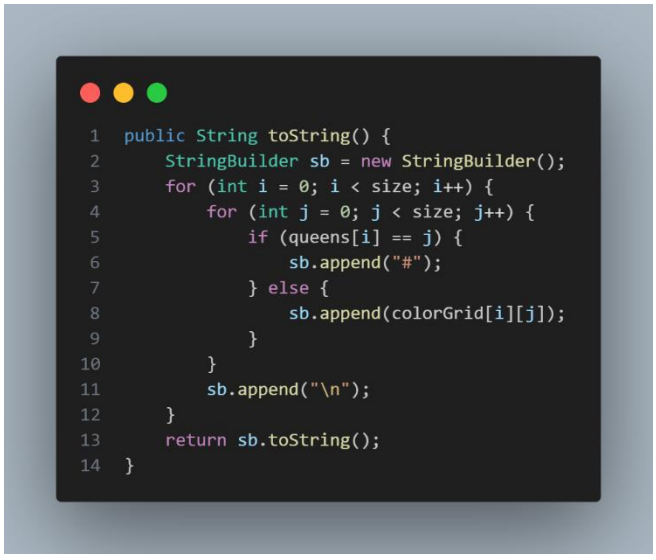
Implementasi pengecekan atau validasi Queen ditulis dalam fungsi `isSafe` yang mengambil suatu koordinat kemudian melakukan *loop* untuk mengecek apakah terdapat Queen pada kolom yang sama, apakah terdapat kesamaan warna, dan apakah Queen bersentuhan dengan Queen lain dengan menghitung jarak mutlak antar *grid*, termasuk diagonal.



```java
public boolean isSafe(int row, int col) {
    char currentColor = getColor(row, col);

    for (int i = 0; i < row; i++) {
        int placedRow = i;
        int placedCol = queens[i];

        // 1. Column Constraint
        if (placedCol == col) {
            return false;
        }

        // 2. Color Constraint
        if (colorGrid[placedRow][placedCol] == currentColor) {
            return false;
        }

        // 3. Touch Constraint
        int rowDiff = Math.abs(placedRow - row);
        int colDiff = Math.abs(placedCol - col);

        if (rowDiff <= 1 && colDiff <= 1) {
            return false;
        }
    }
    return true;
}
```

Gambar 4. Implementasi isSafe

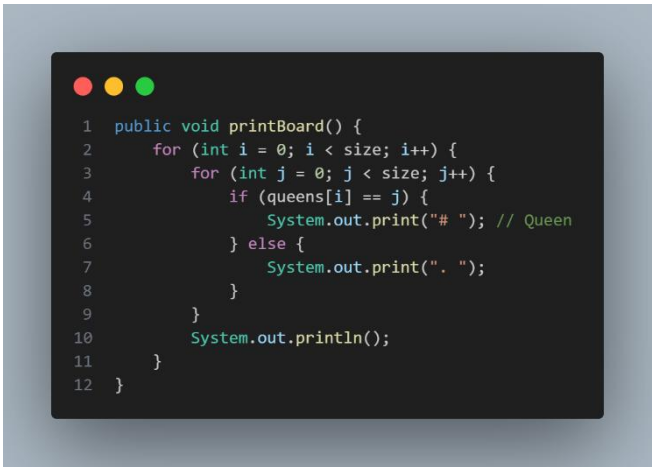
Implementasi `toString` digunakan untuk mengembalikan data dari *grid array* dua dimensi menjadi string untuk ditampilkan pada file solusi. Program ini menggunakan *loop* sederhana dengan memnggunakan "#" apabila terdapat Queen pada koordinat dan kode huruf apabila tidak.



```java
public String toString() {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (queens[i] == j) {
                sb.append("#");
            } else {
                sb.append(colorGrid[i][j]);
            }
        }
        sb.append("\n");
    }
    return sb.toString();
}
```

Gambar 5. Implementasi toString

Implementasi `printBoard` digunakan untuk menuliskan langkah *update* pada terminal. Namun opsi ini dimatikan karena *live update* sudah dilakukan pada GUI.



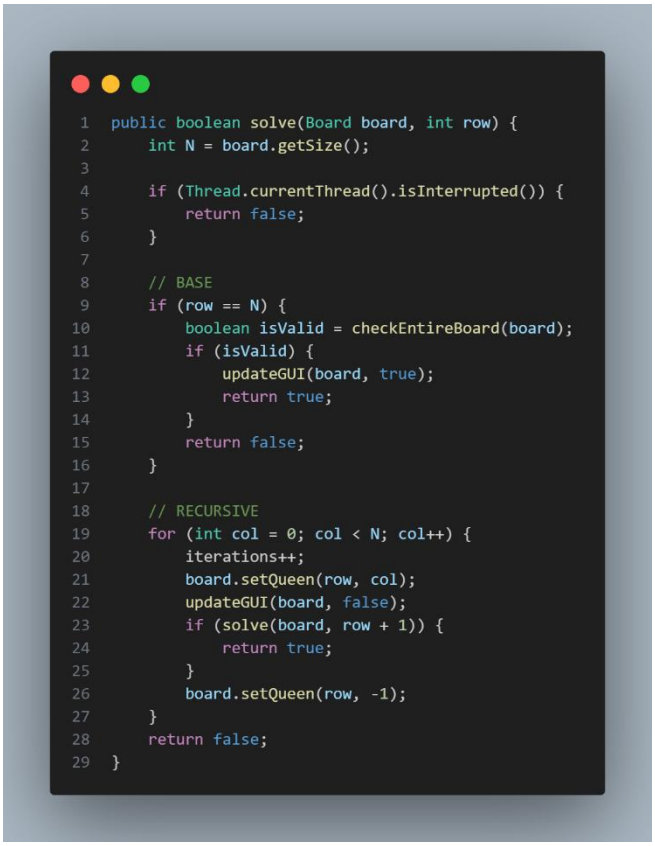
```java
public void printBoard() {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (queens[i] == j) {
                System.out.print("# "); // Queen
            } else {
                System.out.print(". ");
            }
        }
        System.out.println();
    }
}
```

Gambar 6. Implementasi printBoard

C. Kelas Solver

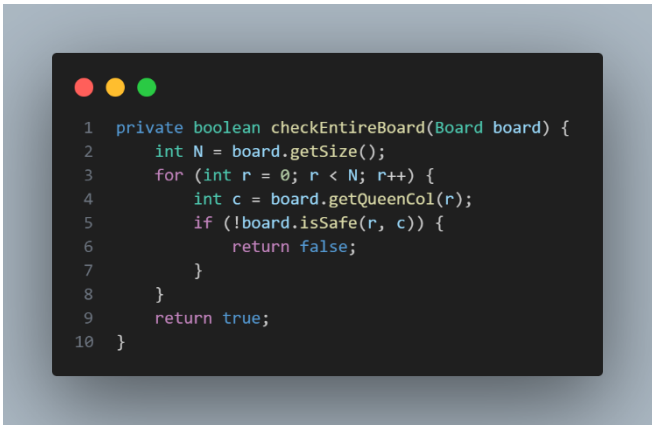
Program akan mengambil tipe data Board, *integer* baris, dan memanggil metode untuk mengambil ukuran *board.* Kemudian program akan dijalankan secara rekursif dengan basis apabila kondisi sudah terpenuhi pada seluruh grid(tidak ada posisi Queen yang melanggar). Kemudian secara rekursif akan dipanggil dengan mengenumerasikan setiap kolom dan meletakkan Queen pada posisi tersebut. Baris-baris dibawahnya akan berjalan dan melalui pengecekan/validasi pada rekursi tersebut.

Apabila tidak memenuhi syarat, program akan backtrack atau kembali ke rekursi di atasnya/baris di atasnya untuk mengenumerasi posisi Queen sebanyak satu langka ke depan. Selain itu, kolom pada Queen akan diset kembali menjadi -1 seperti pada awal inisiasi.



```java
public boolean solve(Board board, int row) {
    int N = board.getSize();

    if (Thread.currentThread().isInterrupted()) {
        return false;
    }

    // BASE
    if (row == N) {
        boolean isValid = checkEntireBoard(board);
        if (isValid) {
            updateGUI(board, true);
            return true;
        }
        return false;
    }

    // RECURSIVE
    for (int col = 0; col < N; col++) {
        iterations++;
        board.setQueen(row, col);
        updateGUI(board, false);
        if (solve(board, row + 1)) {
            return true;
        }
        board.setQueen(row, -1);
    }
    return false;
}
```

Gambar 7. Implementasi Solve



```java
private boolean checkEntireBoard(Board board) {
    int N = board.getSize();
    for (int r = 0; r < N; r++) {
        int c = board.getQueenCol(r);
        if (!board.isSafe(r, c)) {
            return false;
        }
    }
    return true;
}
```

Gambar 8. Implementasi checkEntireBoard

Implementasi `checkEntireBoard` digunakan untuk mengecek keseluruhan *grid* pada *board*. Fungsi ini akan memanggil fungsi `isSafe` secara berulang-ulang untuk mengecek posisi setiap Queen.

```java
private void updateGUI(Board board, boolean forceUpdate) {
    if (listener == null) return;

    long currentTime = System.currentTimeMillis();

    if (forceUpdate || (currentTime - lastUpdateTime >= delayMs)) {

        listener.onUpdate(board, iterations);

        lastUpdateTime = currentTime;
    }
}
```

Gambar 9. Implementasi updateGUI

Seperti namanya, implementasi ini akan memperbarui posisi Queen pada tampilan *board* GUI pada setiap kurun waktu tertentu melalui variabel delayMS.

D. Kelas Main

Program ini digunakan sebagai pintu utama berjalannya program yang akan memanggil Java Swing dan menginstansiasi QueensGUI sekaligus memunculkan antarmuka GUI tersebut pada layar.

```java
public class Main {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            try {
                javax.swing.UIManager.setLookAndFeel(javax.swing.UIManager.getSystemLookAndFeelClassName());
            }
            catch (Exception ignored) { }

            QueensGUI gui = new QueensGUI();
            gui.setVisible(true);
        });
    }
}
```

Gambar 10. Implementasi Main

V. PENGUJIAN

A. Parameter uji dan kasus uji

Untuk memastikan reliabilitas algoritma, simulasi dirancang berdasarkan kasus uji sebagai berikut:

TABEL II. DAFTAR KASUS UJI

Kasus Uji	Ukuran	Nama File
Kasus uji 1	6 × 6	Tc1.txt
Kasus uji 2	7 × 7	Tc2.txt
Kasus uji 3	8 × 8	Tc3.txt
Kasus uji 4	9 × 9	Tc4.txt
Kasus uji 5	10 × 10	Tc5.txt

Isi dari file .txt masing masing kasus uji adalah sebagai berikut,

1) Kasus Uji 1

```
AAABBB
AAABBB
CCCDDD
CCCDDD
EEEFFF
EEEFFF
```

2) Kasus Uji 2

```
AAAAABB
ACCCDBB
ACCEEDD
AFFEEED
FFFFGGG
HHHIIIG
HHHIIIG
```

3) Kasus Uji 3

```
ABBCDDDE
AABCCDDE
FAAAADDE
FFAAAADE
GGAAAADA
HGAAAAAA
HHHAAAAA
HHHHHAAA
```

4) Kasus Uji 4

```
AAABBCCCD
ABBBBCECD
ABBBDCECD
AAABDCCCD
BBBBDDDDD
FGGGDDHDD
FGIGDDHDD
FGIGDDHDD
FGGGDDHHH
```

5) Kasus Uji 5

```
AAABBBCCCC
AABBBCCCCD
ABBBDDDEED
FFBDDDEEED
FFBGGGEEEH
FFGGGEEEHH
FFGIIIIHHH
FFGIIIIHHJ
FFFIIIIJJJ
FFFFIIJJJJ
```

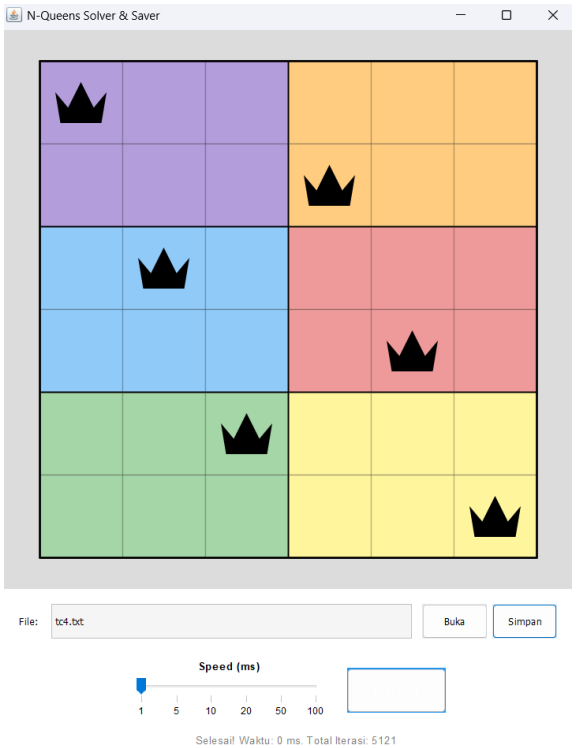
B. Hasil Uji

Hasil dari setiap skenario pengujian adalah sebagai berikut:

1) Hasil uji 1

```
#AABBB
AAA#BB
C#CDDD
CCCD#D
EE#FFF
EEEFF#

Ukuran board: 6x6
Waktu pencarian: 0 ms
Jumlah iterasi: 5121 kasus
```

Kasus uji 1 merupakan pengujian dengan skala terkecil. Algoritma bekerja secara instan karena jumlah kombinasi kemungkinan posisi Queen pada papan 6x6 yang sangat sedikit sehingga solusi ditemukan dengan sangat cepat. Gambar 11 memperlihatkan posisi Queen yang sesuai.

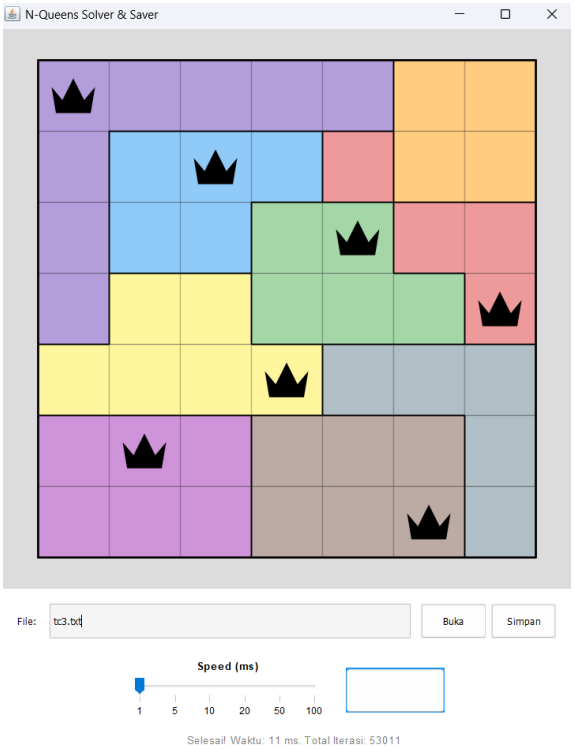


Gambar 11. Tampilan Uji 1

2) Hasil uji 2

```
#AAAABB
AC#CDBB
ACCE#DD
AFFEEE#
FFF#GGG
H#HIIIG
HHHII#G

Ukuran board: 7x7
Waktu pencarian: 11 ms
Jumlah iterasi: 53011 kasus
```

Pada ukuran 7x7, terlihat peningkatan jumlah iterasi yang signifikan yaitu lebih dari 10x lipat dari 6x6, dengan waktu pencarian yang masih sangat singkat yaitu 11 ms. Hal ini menunjukkan bahwa algoritma *brute force* masih sangat efisien untuk menangani *board* ukuran kecil hingga menengah. Gambar 12 memperlihatkan posisi Queen yang sesuai

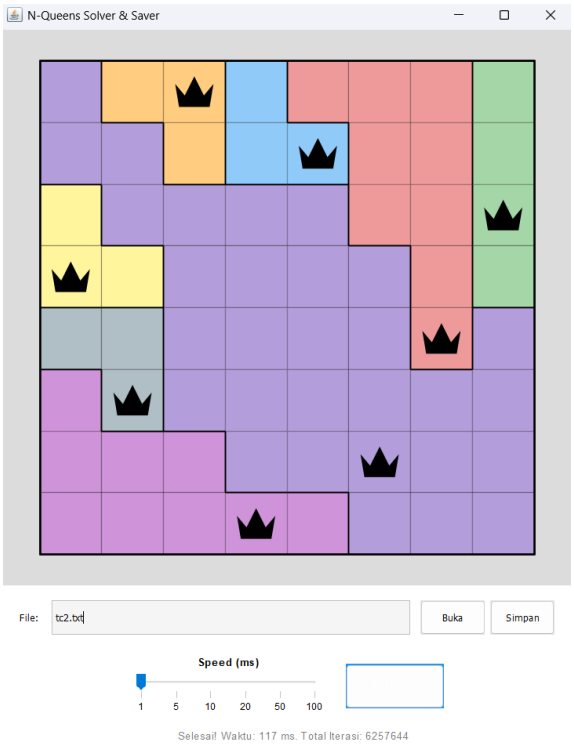


Gambar 12. Tampilan Uji 2

3) Hasil uji 3

```
AB#CDDDE
AABC#DDE
FAAAADD#
#FAAAADE
GGAAAA#A
H#AAAAAA
HHHAA#AA
HHH#HAAA

Ukuran board: 8x8
Waktu pencarian: 117 ms
Jumlah iterasi: 6257644 kasus
```

Terdapat peningkatan waktu dan jumlah iterasi yang signifikan walaupun program dapat memperoleh solusi dalam waktu yang singkat yaitu 117 ms. Gambar 13 memperlihatkan posisi Queen yang sesuai.

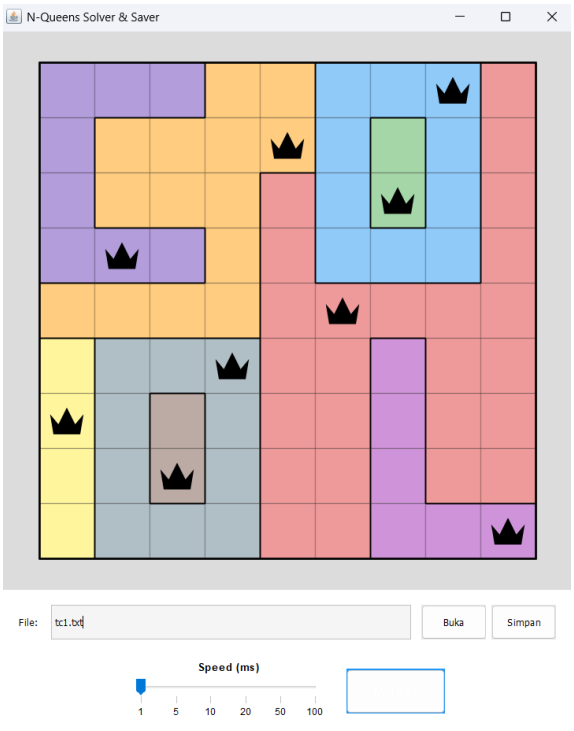


Gambar 13. Tampilan Uji 3

4) Hasil uji 4

```
AAABBCC#D
ABBB#CECD
ABBBDC#CD
A#ABDCCCD
BBBBD#DDD
FGG#DDHDD
#GIGDDHDD
FG#GDDHDD
FGGGDDHH#

Ukuran board: 9x9
Waktu pencarian: 5838 ms
Jumlah iterasi: 364209345 kasus
```

Pengujian menunjukan peningkatan kompleksitas yang sangat signifikan. Dengan lebih dari 364 juta iterasi program membutuhkan waktu sekitar 5,8 detik untuk mendapatkan solusi. Gambar 14 memperlihatkan posisi Queen yang sesuai.

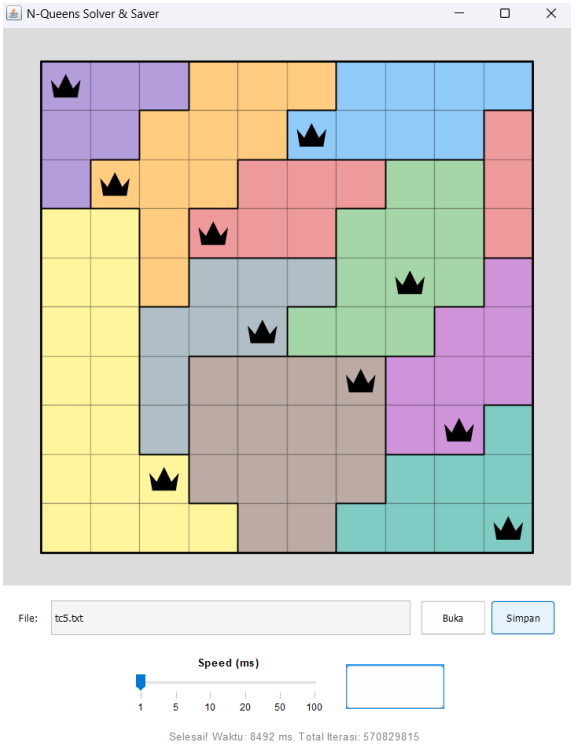


Gambar 14. Tampilan Uji 4

5) Hasil uji 5

```
#AABBBCCCC
AABBB#CCCD
A#BBDDDEED
FFB#DDEEED
FFBGGGE#EH
FFGG#EEEHH
FFGIII#HHH
FFGIIIIH#J
FF#IIIIJJJ
FFFFIIJJJ#

Ukuran board: 10x10
Waktu pencarian: 8492 ms
Jumlah iterasi: 570829815 kasus
```

Pengujian yang dilakukan pada ukuran 10x10 ini tidak memberikan peningkatan waktu dan jumlah iterasi yang signifikan. Hal ini dikarenakan program dibuat untuk langsung berhenti setelah mencari solusi pertama yang mungkin yang dipengaruhi oleh faktor *initial state* dari *board*. Artinya, solusi diperoleh tergantung dengan kerumitan *board.* Dalam kasus ini, posisi Queen pada baris pertama terletak pada kolom pertama sehingga memangkas waktu program untuk mengecek sembilan kolom berikutnya. Gambar 15 menunjukkan posisi Queen yang sesuai.



Gambar 15. Tampilan Uji 5

LAMPIRAN

- GitHub Repository:
- Tabel spesifikasi:

No	Poin	Ya	Tidak
1	Program berhasil di kompilasi tanpa kesalahan	✓	
2	Program berhasil di jalankan	✓	
3	Solusi yang diberikan program benar dan mematuhi aturan permainan	✓	
4	Program dapat membaca masukan berkas .txt serta menyimpan solusi dalam berkas .txt	✓	
5	Program memiliki Graphical User Interface (GUI)	✓	
6	Program dapat menyimpan solusi dalam bentuk file gambar		✓

REFERENSI

[1] M. Dr. Ir. Rinaldi, "02-Algoritma-Brute-Force-(2026)-Bag1," 2026. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/02-Algoritma-Brute-Force-(2026)-Bag1.pdf.

[2] M. Dr. Ir. Rinaldi, "03-Algoritma-Brute-Force-(2026)-Bag2," 2026. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/03-Algoritma-Brute-Force-(2026)-Bag2.pdf.

[3] Geeks for Geeks, "Brute Force Approach and its pros and cons," 23 July 2025. [Online]. Available: https://www.geeksforgeeks.org/dsa/brute-force-approach-and-its-pros-and-cons/.

[4] Geeks for Geeks, "N Queen Problem," 27 September 2025. [Online]. Available: https://www.geeksforgeeks.org/dsa/n-queen-problem-backtracking-3/.

[5] LinkedIn, "Queens, a puzzle by LinkedIn," [Online]. Available: https://www.linkedin.com/showcase/queens-game/posts/?feedView=all.

PERNYATAAN

Tugas ini disusun sepenuhnya tanpa bantuan kecerdasan buatan (Generative AI), melainkan hasil pemikiran dan analisis mandiri.

Bandung, 18 Februari 2026

Moh. Hafizh Irham Perdana (13524025)